Experiment:1.2

**Aim:**

Write a program to assess various feature matching algorithms for object recognition.

**Software Required:**

Any IDE (Jupyter Notebook, Pycharm, Google Colab).

**Description:**

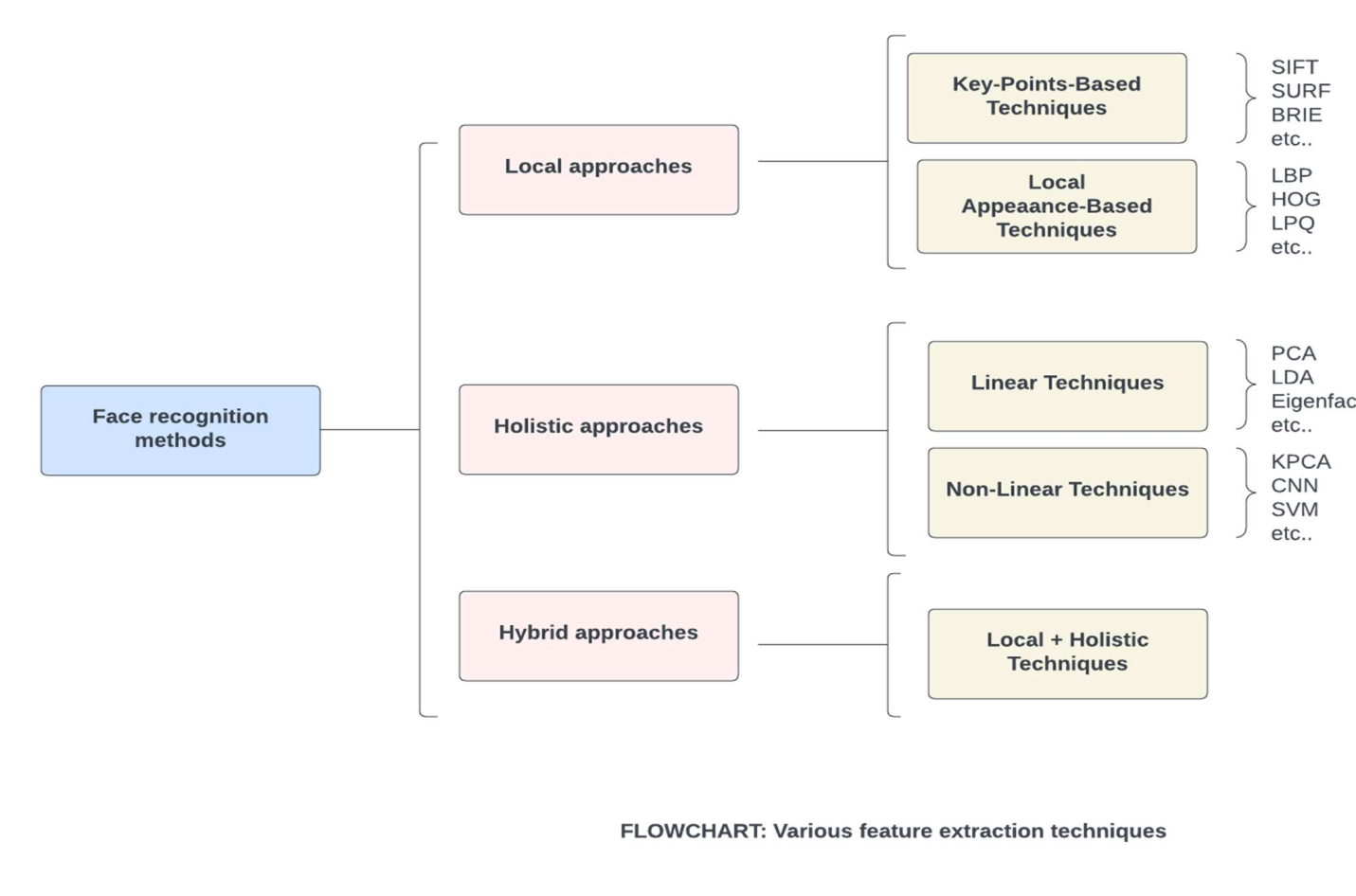
Feature matching algorithms are crucial for object recognition tasks as they establish correspondences between keypoints or descriptors extracted from images. Some commonly used feature matching algorithms for object recognition are:

* Brute-Force Matching: Brute-force matching involves comparing each feature in one image with every feature in another image using a distance metric (e.g., Euclidean distance, Hamming distance). It is a straightforward approach but can be computationally expensive for large feature sets.
* Fast Library for Approximate Nearest Neighbors (FLANN): FLANN is an approximate nearest neighbor search library that accelerates feature matching by using indexing structures. It performs fast approximate nearest neighbor searches, reducing the computational cost compared to brute-force matching.
* Nearest Neighbor with Ratio Test: In this approach, the nearest neighbors of each feature are found, and a ratio test is applied to select the best match. The ratio test compares the distances between the best match and the second-best match. If the ratio is below a certain threshold, the match is considered valid. This method helps to reduce false matches caused by ambiguous correspondences.
* Random Sample Consensus (RANSAC): RANSAC is an iterative algorithm used to estimate the transformation model between two sets of matched keypoints. It is commonly used in object recognition tasks to robustly estimate the pose or alignment of an object in a scene. RANSAC helps eliminate outliers and find a reliable transformation model.
* Geometric Verification: Geometric verification techniques verify the spatial consistency of matched keypoints by applying geometric constraints. Examples include checking the number of inliers in a homography matrix estimation or the number of correspondences consistent with an affine transformation.
* Graph-based Methods: Graph-based approaches model the feature matches as nodes in a graph, and edges represent the pairwise similarities between the features. Graph matching algorithms, such as graph cuts or spectral clustering, can be applied to find the best correspondences by optimizing a certain objective function.
* Binary Descriptor Matching: For binary descriptors like Binary Robust Invariant Scalable Keypoints (BRISK) or ORB (Oriented FAST and Rotated BRIEF), specialized matching techniques like Hamming distance-based matching or bitwise operations can be used to efficiently match descriptors.
* Deep Learning-based Matching: With the advent of deep learning, neural networks can be trained to directly predict correspondences between image patches or descriptors. Siamese networks or matching networks can be used to learn the matching function, allowing for end-to-end feature matching.

**Steps:**

1. Import the necessary libraries and modules (e.g., OpenCV).
2. Load the reference image and the query image.
3. Extract features from both images using a chosen feature extraction technique (e.g., SIFT, SURF, ORB).
4. Apply feature matching algorithms (e.g., Brute-Force Matcher, FLANN) to match the extracted features.
5. Compute the similarity or matching score between the reference image and the query image based on the matched features.
6. Evaluate and compare the performance of different feature matching algorithms using suitable metrics (e.g., accuracy, precision, recall, F1-score).
7. Analyze and interpret the results to identify the strengths and weaknesses of each algorithm.
8. Repeat the experiment with different combinations of feature extraction and matching algorithms to gain further insights.
9. Document the findings, observations, and conclusions.

**Pseudo code/Algorithms/Flowchart/Steps:**



**Implementation:**

from google.colab import drive

drive.mount('/content/gdrive')

"""\*\*Import Libraries\*\*"""

import cv2 as cv

import skimage.io as io

"""\*\*Load Images\*\*"""

image1 = cv.imread("/content/gdrive/MyDrive/Data/images/tajmahal.png", cv.IMREAD\_GRAYSCALE)

image2 = cv.imread("/content/gdrive/MyDrive/Data/images/tajmahal1.png", cv.IMREAD\_GRAYSCALE)

"""\*\*Innitialize keypoints detector and descriptor\*\*"""

obr = cv.ORB\_create()

cv.SIFT\_create()

"""\*\*Detect keypoints detector and descriptors\*\*"""

keyPoint1, desc1 = obr.detectAndCompute(image1,None)

keyPoint2, desc2 = obr.detectAndCompute(image2,None)

"""\*\*Innitialize Brute Force Matcher\*\*"""

bf=cv.BFMatcher\_create()

"""\*\*Match the Descriptor using Brute Force\*\*"""

matches = bf.match(desc1,desc2)

"""\*\*Sort the Matches by Distance\*\*"""

matches = sorted(matches,key=lambda x: x.distance)

"""\*\*Draw top 'n' matches\*\*"""

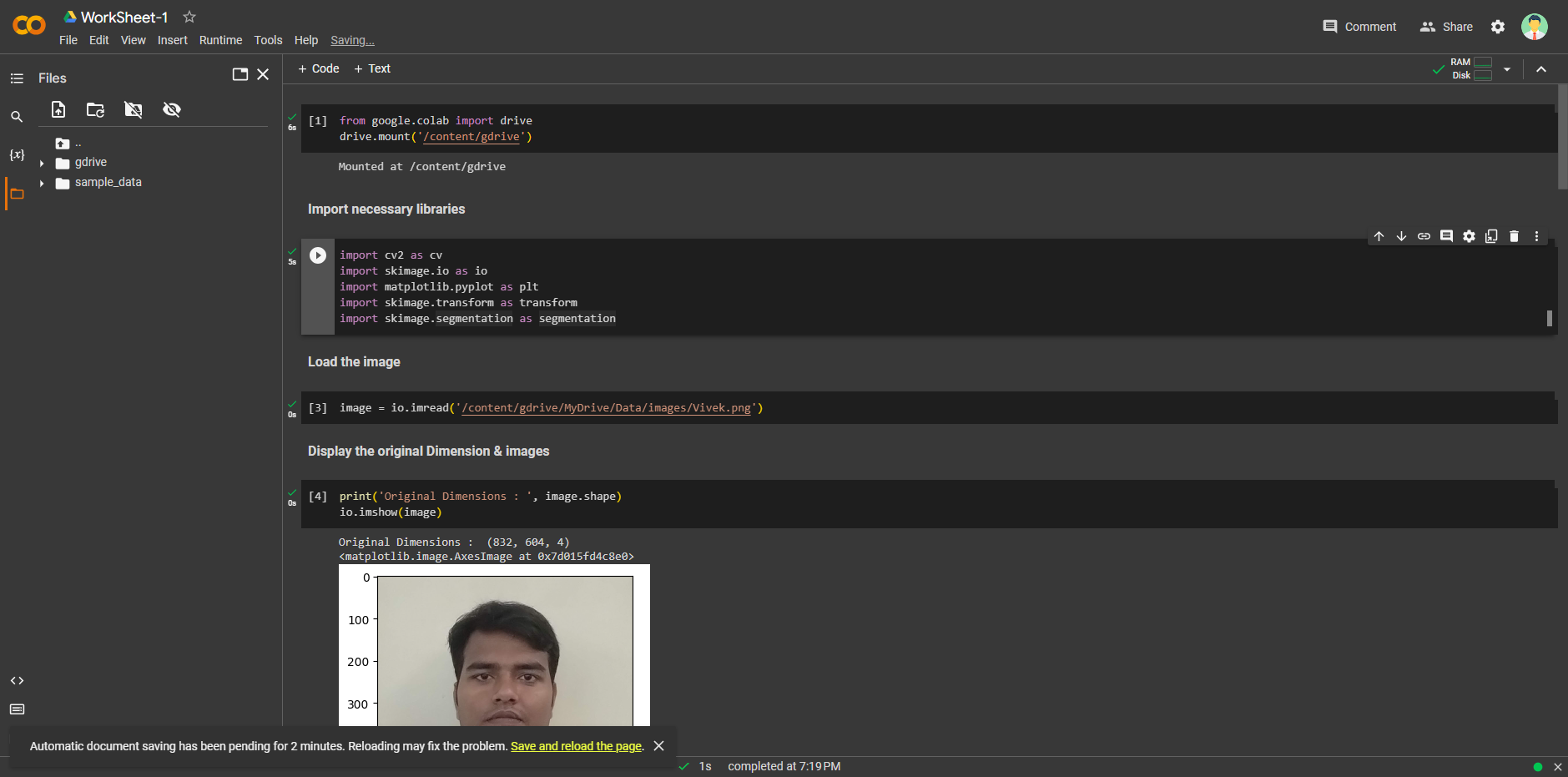
n=20

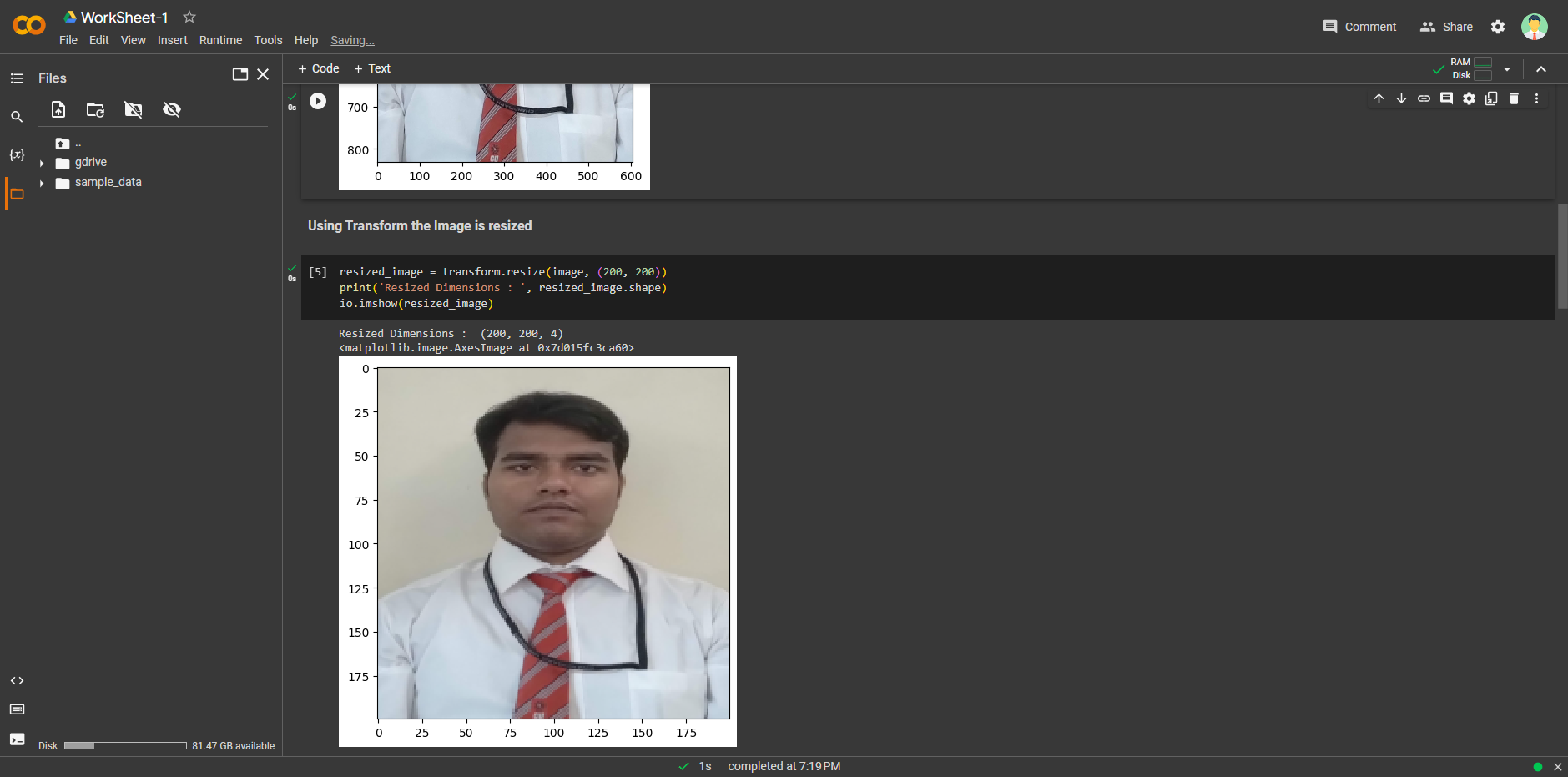
result = cv.drawMatches(image1,keyPoint1,image2,keyPoint2,matches[:n],None,flags=2)

"""\*\*Display Result\*\*"""

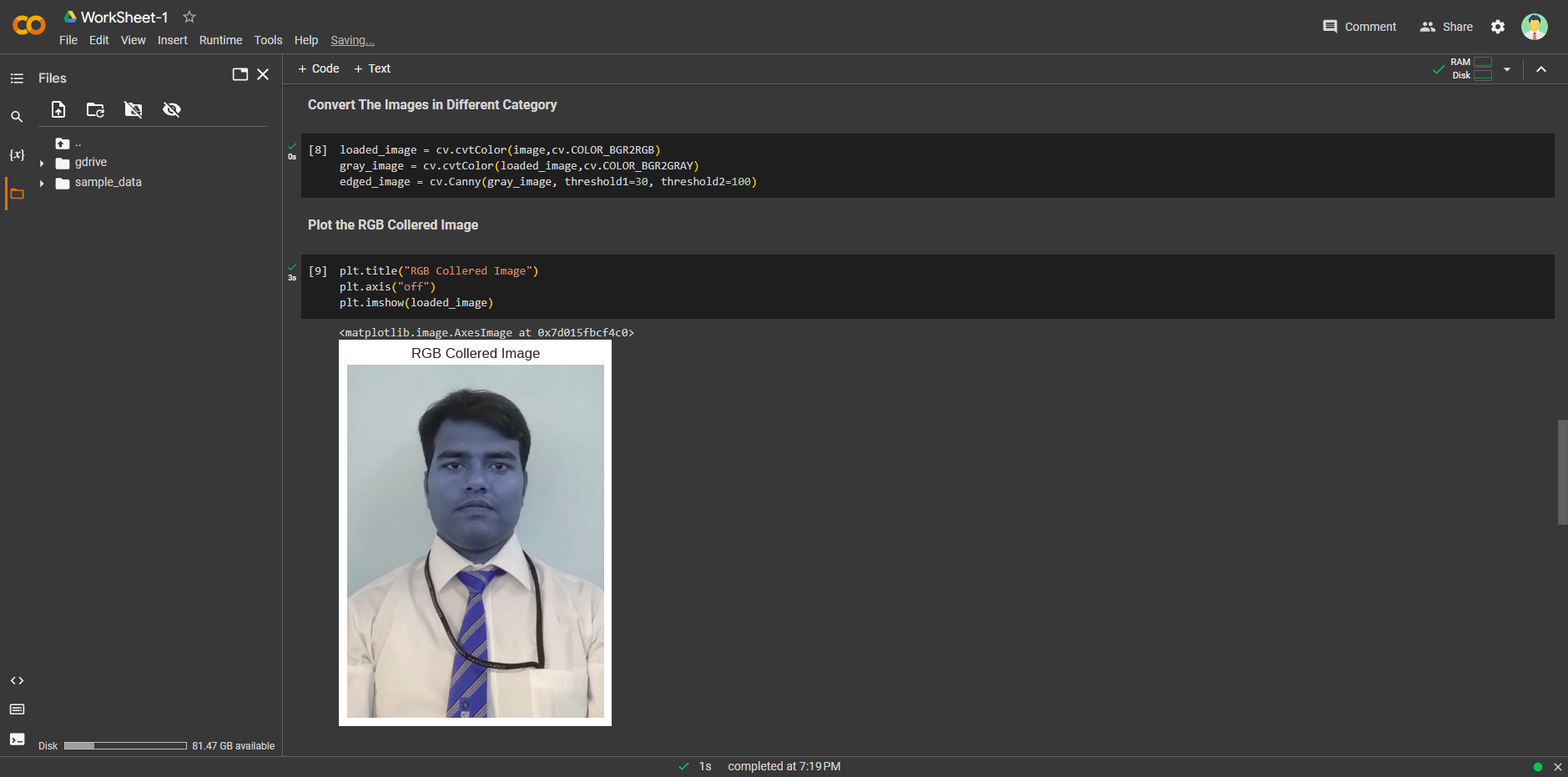
io.imshow(result)

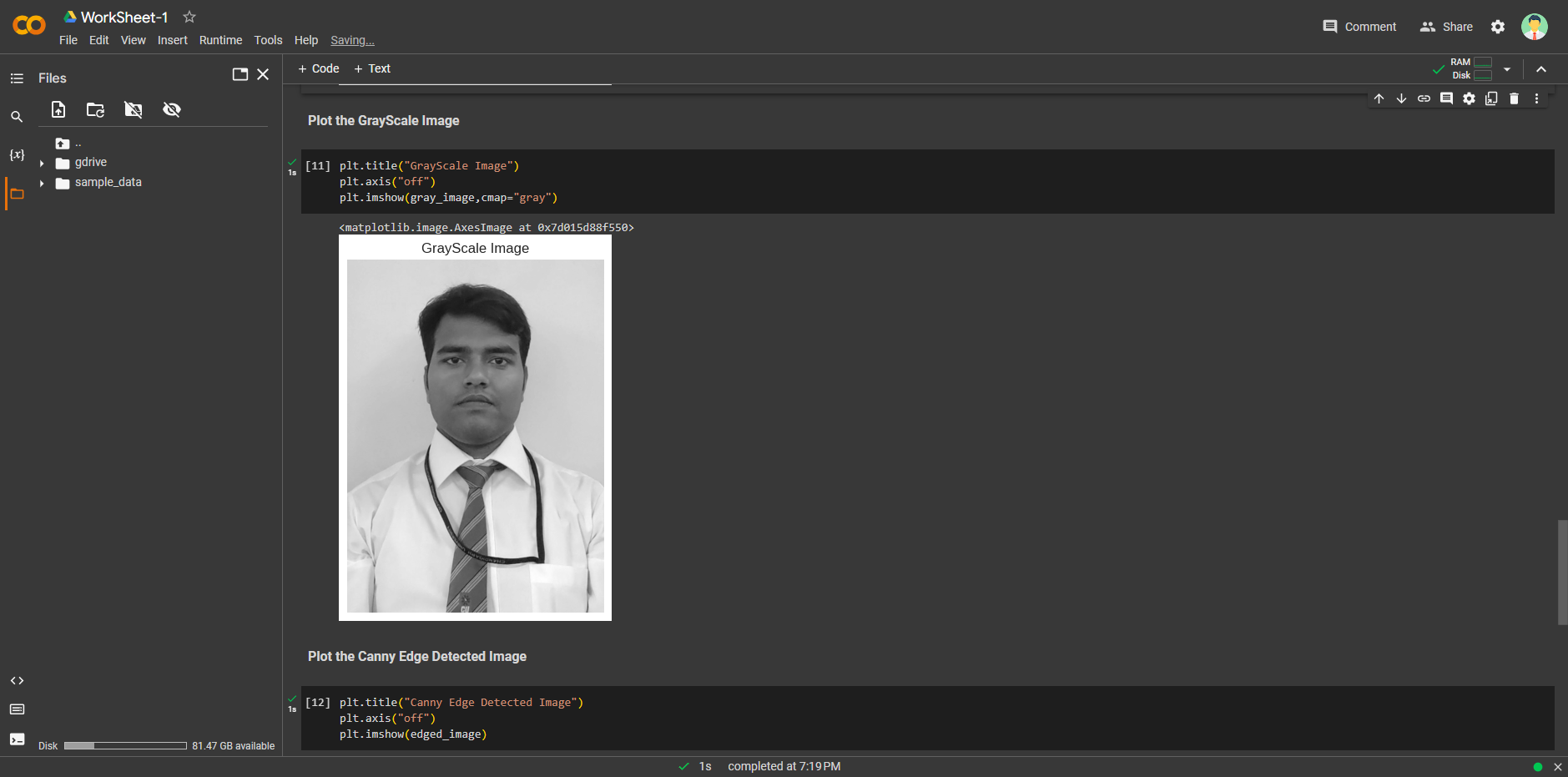
**Output:**

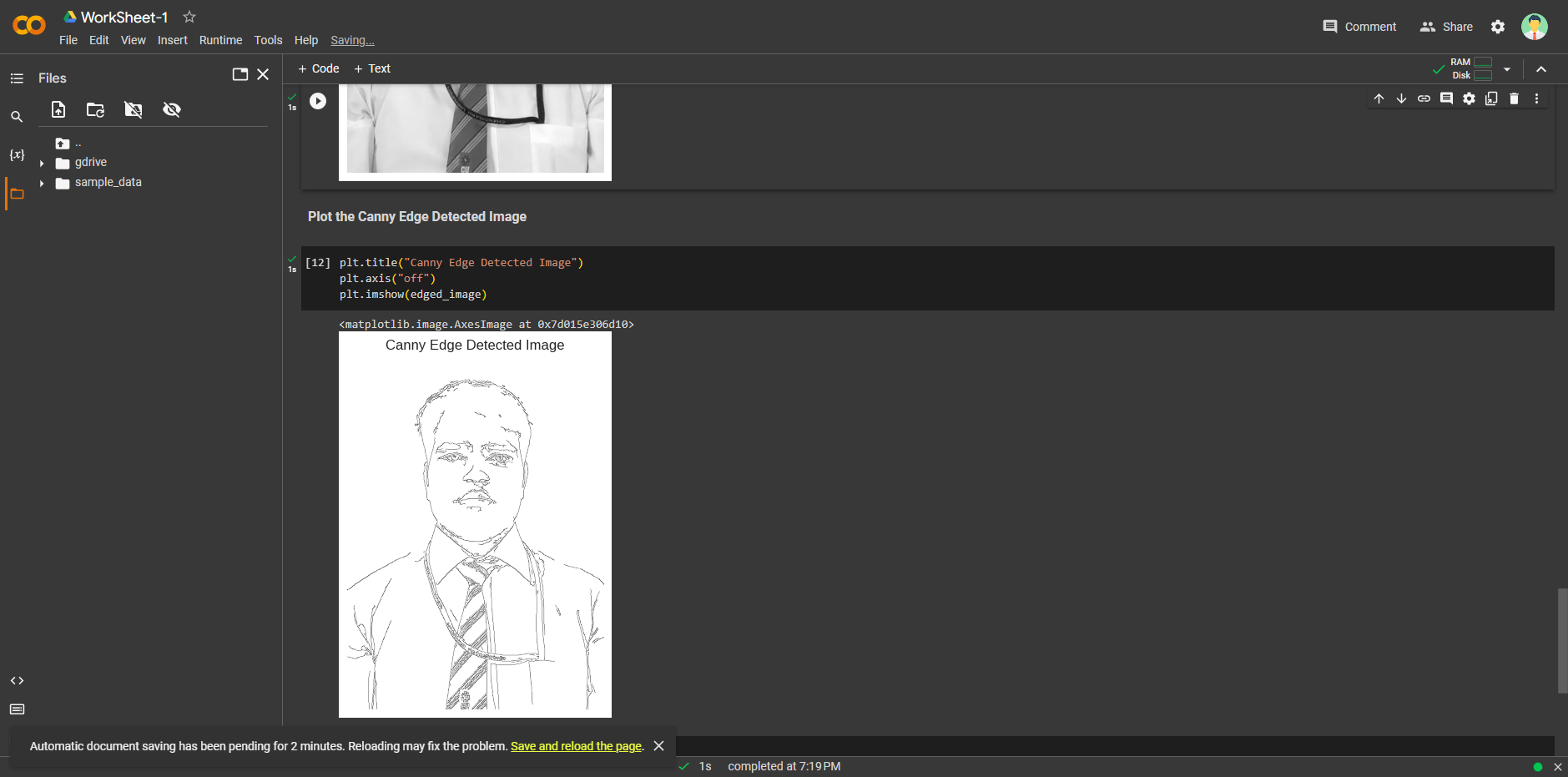
****

****

****

****

****

****